NASA/CR—1998-208684

/N-31
432 203
20p

Mechanical System Analysis/Design Tool (MSAT) Quick Guide

HauHua Lee, Mark Kolb, and Jack Madelone
GE Corporate Research and Development Center, Schenectady, New York

November 1998

NASA/CR—1998-208684

Mechanical System Analysis/Design Tool (MSAT) Quick Guide

HauHua Lee, Mark Kolb, and Jack Madelone
GE Corporate Research and Development Center, Schenectady, New York

Prepared under Contract NAS3–26617

National Aeronautics and
Space Administration

Lewis Research Center

November 1998

Mechanical System Design/Analysis Tool (MSAT)

Introduction

MSAT is a unique multi-component multi-disciplinary tool that organizes design analysis tasks around object-oriented representations of *configuration components*, *analysis programs* and *modules*, and *data transfer links* between them (Figure 1). This creative modular architecture enables rapid generation of input stream for trade-off studies of various engine configurations. The *data transfer links* automatically transport output from one application as relevant input to the next application once the sequence is set up by the user. The computations are managed via *constraint propagation* - the constraints supplied by the user as part of any optimization module. The software can be used in the preliminary design stage as well as during the detail design of product development process.

This software has been used in HSCT program to optimize the design of Exhaust Nozzle. It has also been used in design of JSF lift and main engines and GE90 engine. It will be integrated with NASA's Numerical Propulsion System Simulation (NPSS) that is available to the US aeronautics community, as well as used internally at NASA for coupling conceptual and preliminary design codes for propulsion and propulsion/airframe system analysis. MSAT may be used for design and analysis of any mechanical design where a number of applications from different disciplines are used in simulating a component or a system with multi-component assembly. MSAT has also been integrated with modules such as *Monte Carlo*, *Design of Experiments*, *Response Surfaces*, *Optimization* to provide robust design and uncertainty analysis capability in preliminary design. This added capability in MSAT identifies whether the product is under-designed (there is a risk) or over-designed (it costs more than necessary).

MSAT software provides global perspective on system design. The plug-and-play framework enables the user to add new applications and/or components and perform quick trade-off studies. This inherent capability is key to "quality" design since 80% of the cost of the product gets locked-in during the initial 20% effort. In addition, MSAT fulfills the critical requirement of providing zooming capability required in NPSS environment. A user can conveniently move from 1-D to 2-D to 3-D using the same user-interface and same tool-set.

Because MSAT framework allows easy extension by adding new modules it can be continuously improved to become more versatile by plugging in new optimization and robust design modules without extensive effort. As new advanced software are developed, a user can quickly plug these in the MSAT environment without throwing away the old pieces. This building-block approach will provide tremendous cost benefits to the developers and designers alike.

The following MSAT Quick Guide provides a condensed description of the MSAT capability. The purpose of this manual is to enable new MSAT users to become familiar with the tool and to begin creating MSAT models with minimal time and effort. A complete and detailed MSAT User's Guide is available on request.

Chuck Lawrence
NASA Lewis Research Center

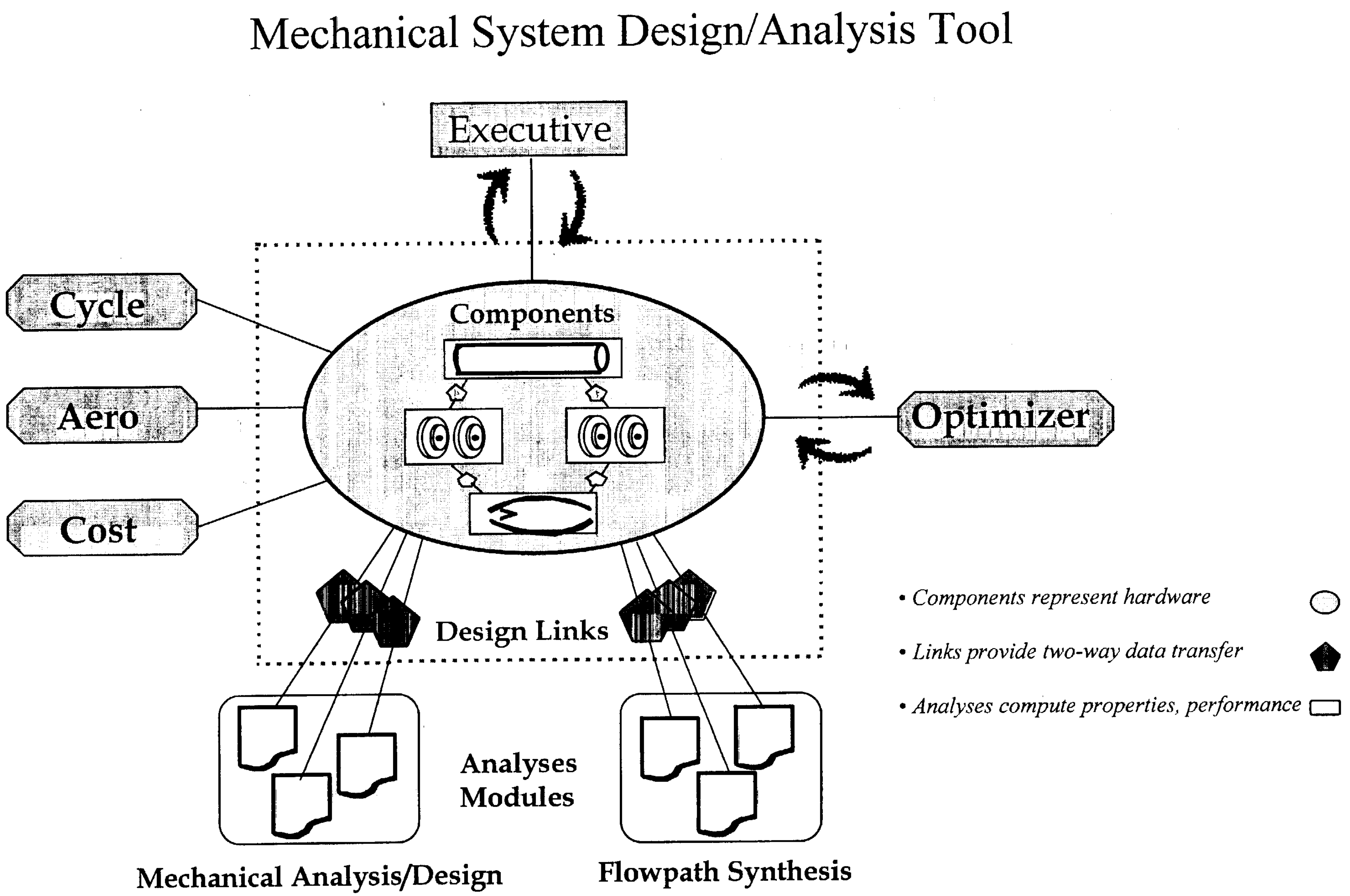

Mechanical System Design/Analysis Tool
Executive
Cycle
Aero
Cost
Components
Optimizer
Design Links
Analyses
Modules
Mechanical Analysis/Design
Flowpath Synthesis
• Components represent hardware
• Links provide two-way data transfer
• Analyses compute properties, performance

Figure 1: The Architecture of MSAT Software

Mechanical System Design/Analysis Tool

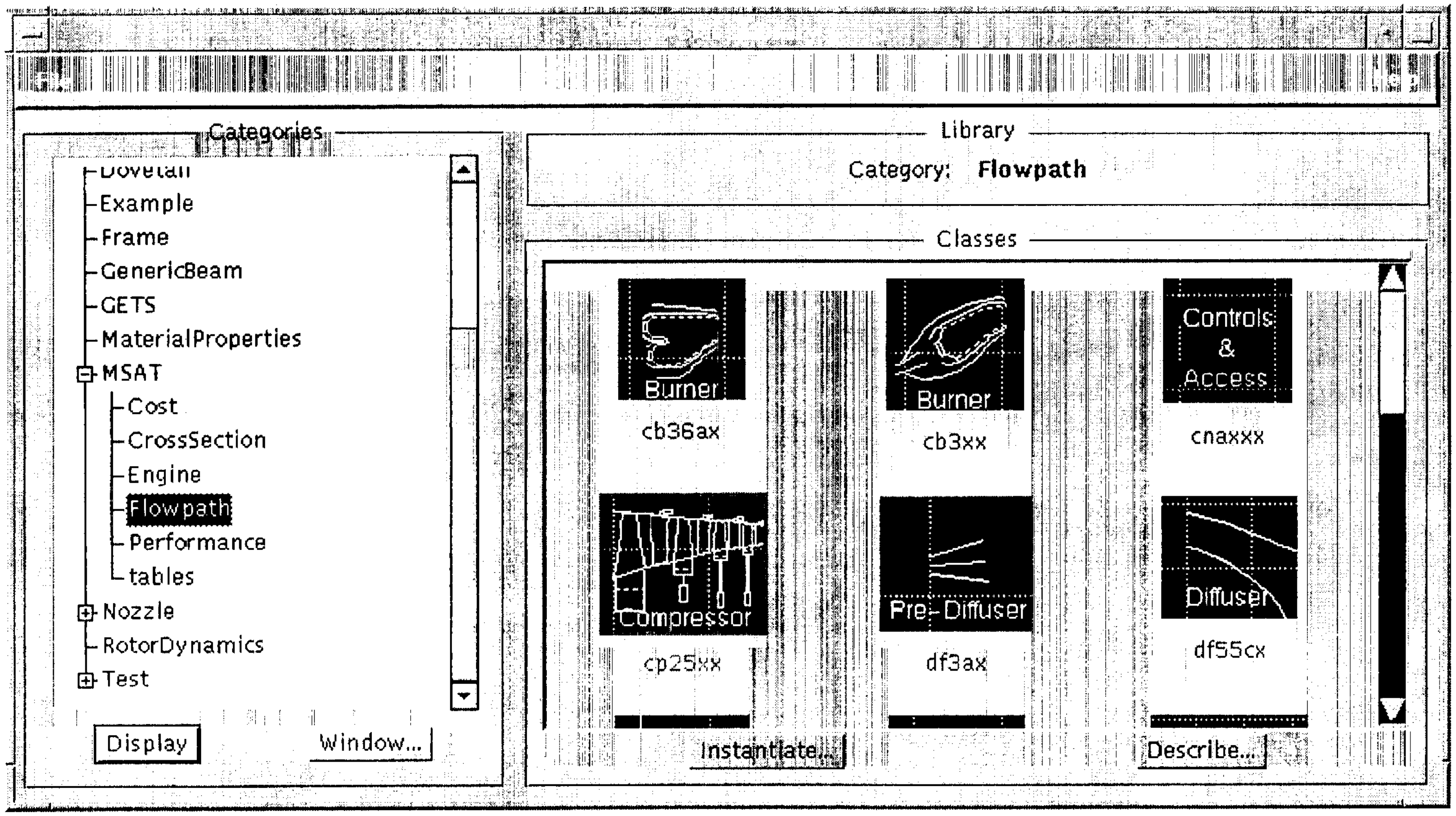


Figure 2: Icon-Based Library Browser for Creating Design Models

Mechanical System Analysis/Design Tool (MSAT) Quick Guide

HauHua Lee
Mark Kolb
Jack Madelone
Engineering Mechanics Laboratory
GE Corporate Research & Development Center
Schenectady, NY
Feb. 1998

This quick guide provides a concise description about the use of MSAT system. MSAT is an object-oriented modeling package for representing engineering artifacts in terms of attributes and constraints on those attributes. The physical objects comprising a system are modeled as **component** objects. System-wide properties (e.g., total weight) are defined using **ensemble** objects. The analyses applied to these components to determine their properties and performance are modeled as **program** and **module** objects. To permit maximum flexibility and modularity, components are not directly associated with their analyses, but instead communicate via intermediate **link** objects. This makes it easy to add new analyses, and to switch to more sophisticated analyses as the design progresses. Designs are modeled by creating instances of the appropriate component, ensemble, program, module, and link classes. Constraint propagation manages the flow of data among the instances.

This document contains two parts. Part A presents major system features for end users. Part B illustrates an example of application development processes. In Appendix, a general comparison between MSAT and other similar systems is provided.

Part A : Running MSAT

Step 1. Starting MSAT GUI

Figure A.1 shows the MSAT's top level window, invoked by entering "client" command at UNIX level. Click on "Server -> Start Server" to initiate the MSAT kernal process.

Step 2. Creating MSAT Models

Figure A.2 shows the class library window, which displays entities predefined by application developers. To create a new model from this library, first declare a new model by clicking on "File -> New" on Top-Level window and a model window with empty canvas pops up, then select the entity and click on "instantiate" - the instantiated entities then show up in the model window canvas, as shown in Figure A.3. Note that the graphical connections among entities are established via the instantiation of link entities.

To view the content of a model, first select an entity on the model canvas, then click on "Display", then an entity window pops up, as shown in Figure A.4. At present, only parameters (with values) can be viewed. To find out other detailed information about entities, one needs to refer to the entity files.

Step 3. Executing MSAT Models

The execution of MSAT models is invoked by changing parameter value in the entity window and then click on "Submit Changes". The model window does not show the execution sequence explicitly. To find out such sequence information, one should monitor the model window where entities being executed are highlighted.

Step 4. Load/Save MSAT Models

You can load an existing model file (*.model) or save the current model to a file for future restoration. The "File" button on the top level window provides the access to these functions.

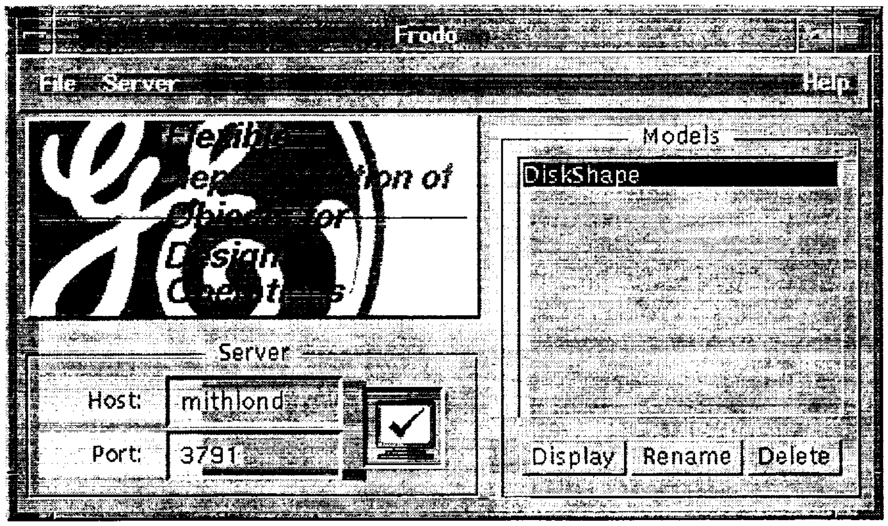


Figure A.1. Top Level Window

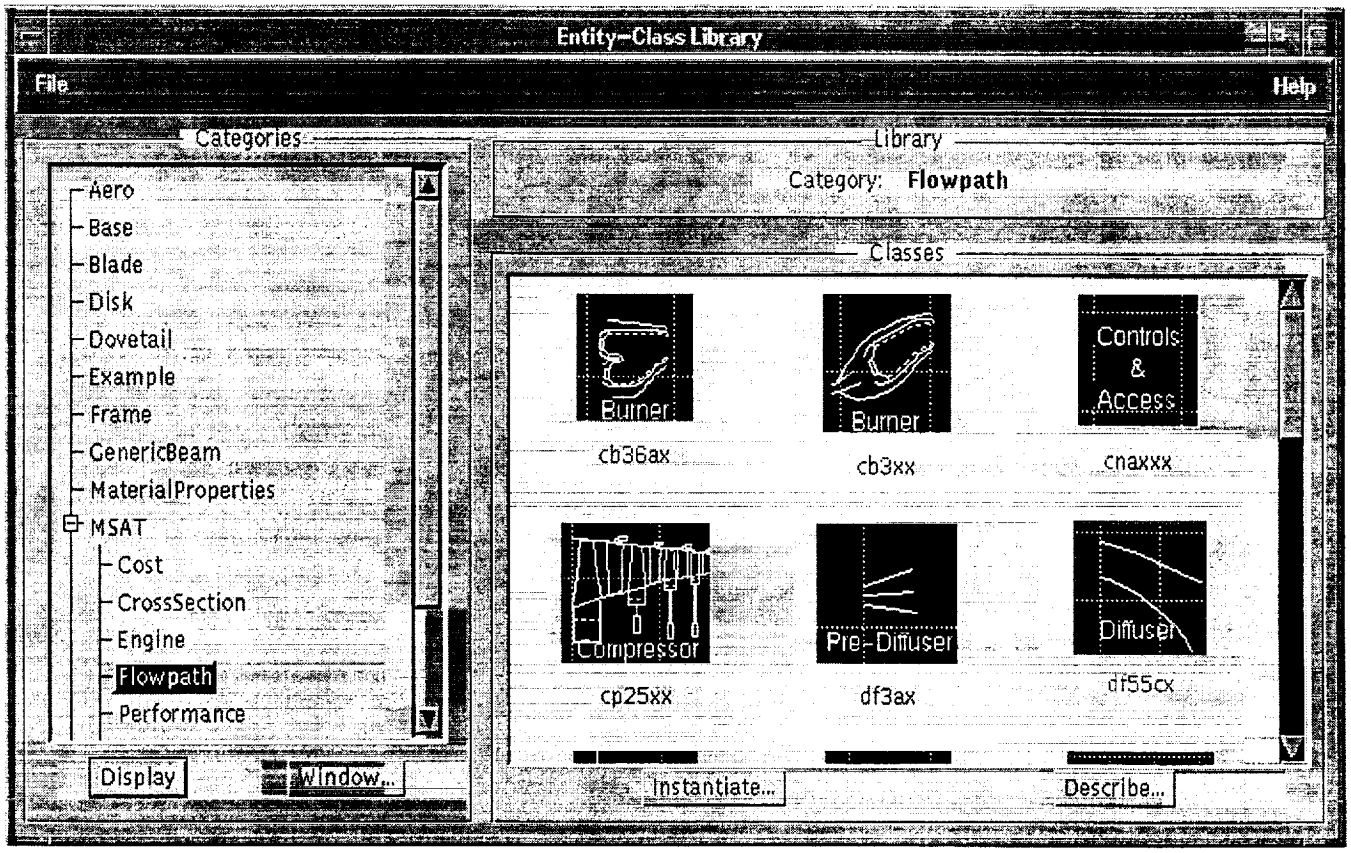


Figure A.2. Class Library Window

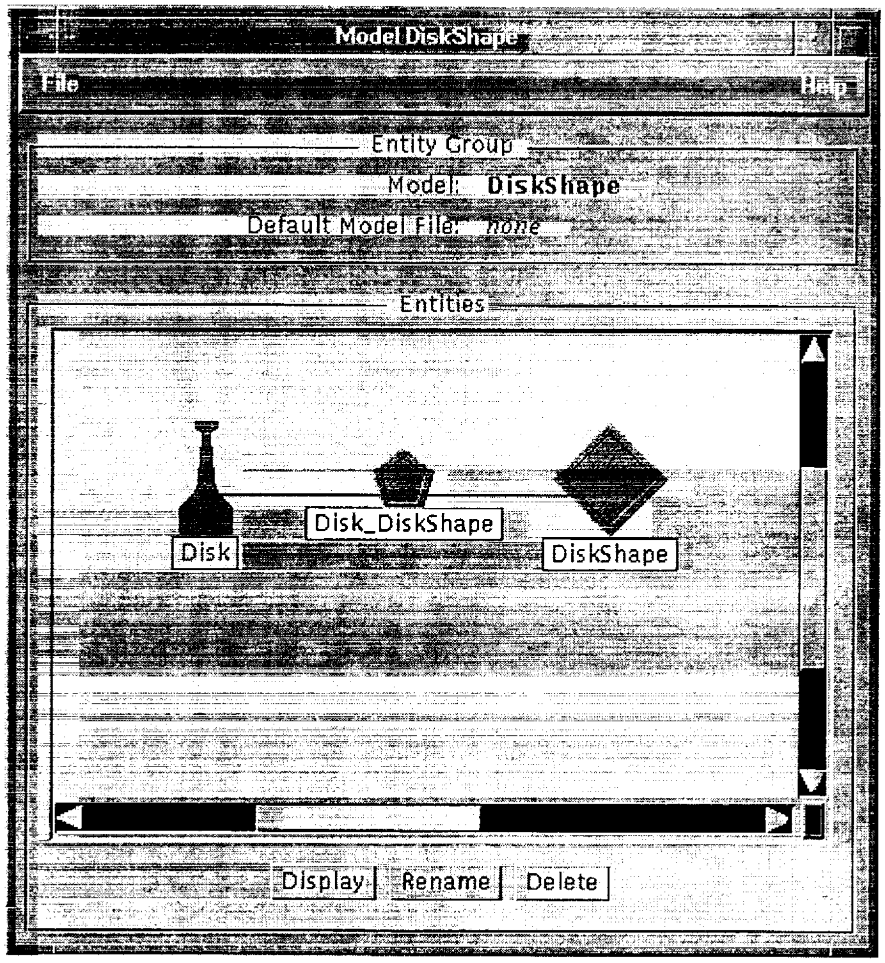
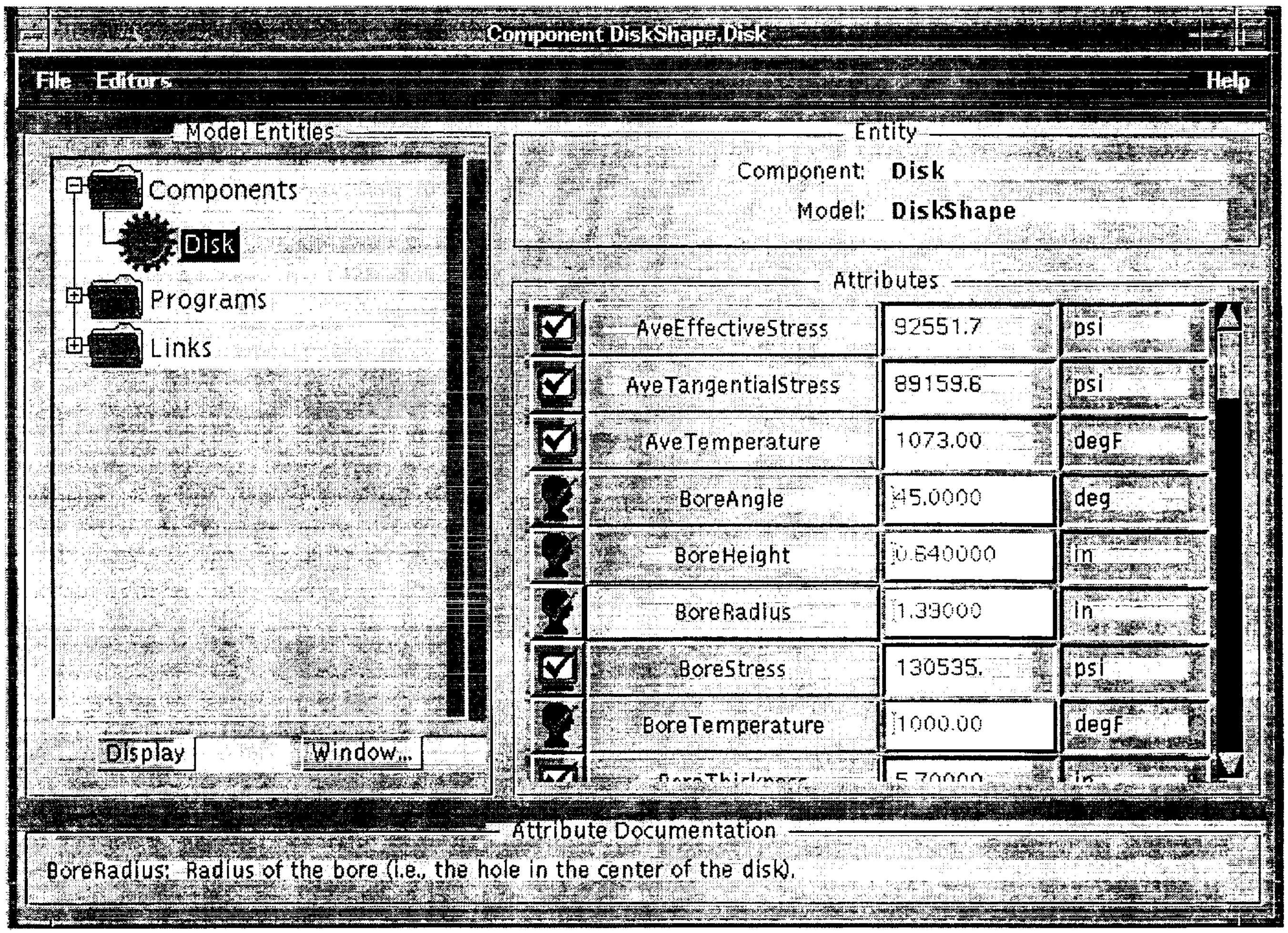


Figure A.3. Model Window



Component: **Disk**
Model: **DiskShape**

✓	AveEffectiveStress	92551.7	psi
✓	AveTangentialStress	89159.6	psi
✓	AveTemperature	1073.00	degF
	BoreAngle	45.0000	deg
	BoreHeight	0.640000	in
	BoreRadius	1.39000	in
✓	BoreStress	130535.	psi
	BoreTemperature	1000.00	degF

BoreRadius: Radius of the bore (i.e., the hole in the center of the disk).

Figure A.4. Entity Window

Part B : Developing MSAT Applications

This section uses a simple example to illustrate the MSAT application development processes. The example involves *wrapping up* a small UNIX code, called piston, as a MSAT **Program** entity, then coupling with a trivial **Module** entity (named Demo), using a **Link** entity (named Piston_Demo).

Problem Descriptions

The piston problem is illustrated in Figure B.1: the hydraulic piston has four design variables, named

X1, X2, X3, and X4, and computes the volume of oil required to lift the load from 0 to 45 degrees, as well as design constraints about force equilibrium, G1 and G2, maximum bending moment, G3, and a minimum piston stroke G4. The reference values of the design variables are X1=84.0, X2=60.0, X3=84.0, and X4=6.0. This problem is used in the book "Numerical Optimization Techniques for Engineering Design" by Vanderplaats, G., 1984, McGraw-Hill.

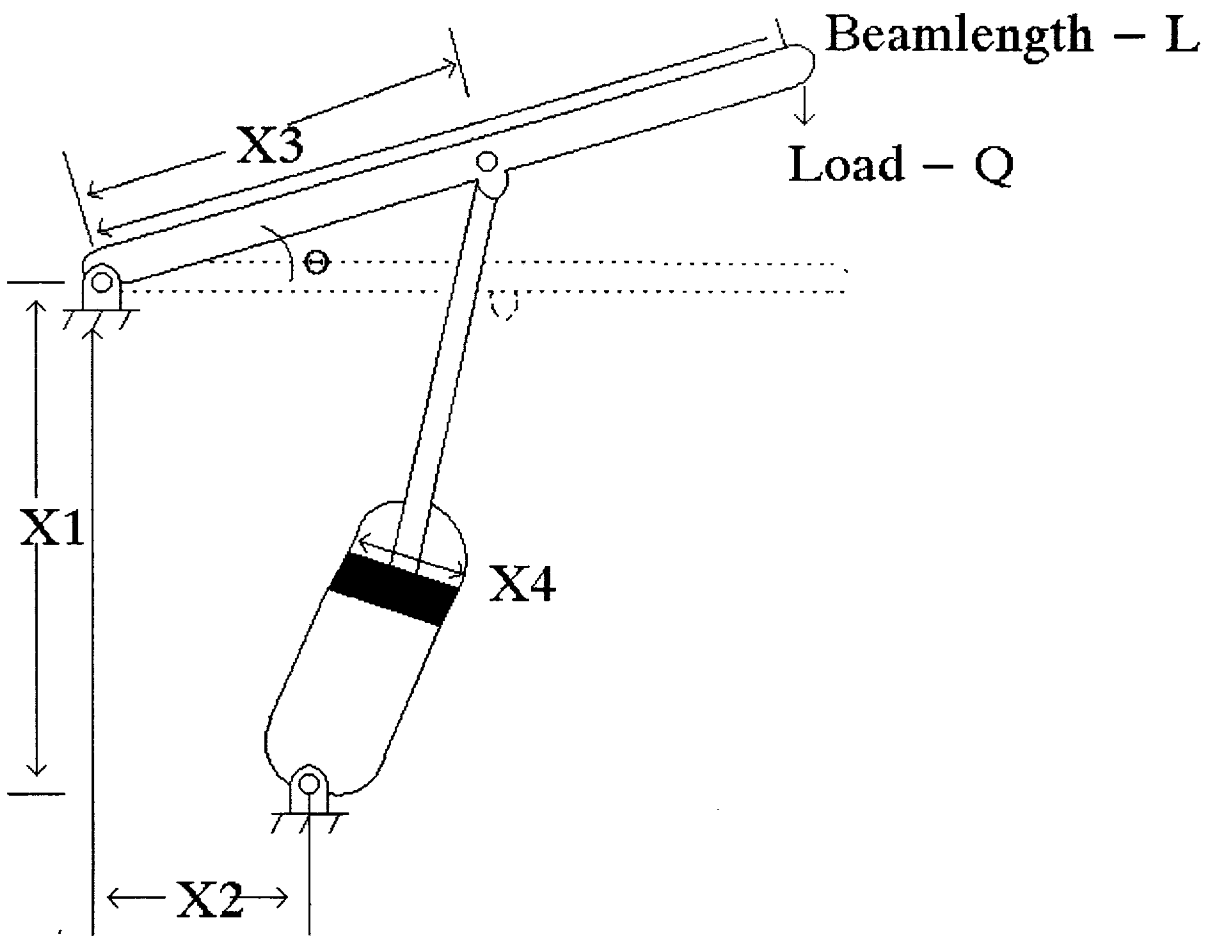


Figure B.1. Piston Example Problem

Files Descriptions

The example directory contains the following files:

- **piston** - Executable.
- **piston.indata** - Sample input file for the executable.
- **piston.outdata** - Sample output file for the executable.

- **Piston.program** - Defines the **program** entity, Piston, wrapping up the piston executable.
- **PistonInputs.inputspec** - Input specification of Piston.program.
- **PistonOutputs.outputspec** - Output specification of Piston.program.

- **inputs.template** - Intermediate file used by PistonInputs.inputspec.
- **outputs.awk** - Intermediate file used by PistonOutputs.outputspec.

- **Demo.module** - Defines the **module** entity, Demo.
- **Piston_Demo.link** - Defines the **link** entity Piston_Demo.
- **piston-link.model** - Defines the model containing the three entities, Piston, Demo and Piston_Demo.

At unix prompt, the piston code can be executed by "piston < piston.indata". The code requires four parameters, defined as X1, X2, X3 and X4. In the input spec of Piston.program, an additional input parameter, Xsum, was created as the sumation of X1, X2, X3 and X4. Xsum was created for the purpose of exercising the constraint propagation. Figure B.2, B.3 and B.4 show the contents of the program definition file and its input/output spec files. The full syntax of these files are accessible from the MSAT on-line help.

```
Program: Piston
RunCommand: "/home/leeh/Frodo-piston/piston < $$FTMPDIR/piston.indata \
                > $$FTMPDIR/piston.outdata"
IOSpecs:
        Inputs isA PistonInputs
        Outputs isA PistonOutputs
EndIOSpecs
Attributes:
    Category: RunCommand
        Integer: ExitStatus
        Default: 0
        EndInteger
    EndCategory
EndAttributes
EndProgram
```

Figure B.2. Piston.program - defining the program entity Piston.

```
InputSpec: PistonInputs
FilterCommand: "$$FBINDIR/ssub \
          -template /home/leeh/Frodo-piston/inputs.template \
           > $$FTMPDIR/piston.indata"

Attributes:
    Category: FilterCommand
        Integer: ExitStatus
        Default: 0
        EndInteger
    EndCategory
        Numeric: X1
        Dom: 10
        EndNumeric
        Numeric: X2
        Dom: 10
        EndNumeric
        Numeric: X3
        Dom: 10
        EndNumeric
        Numeric: X4
        Dom: 10
        EndNumeric

        Numeric: Xsum
        Dom: 100
        EndNumeric
EndAttributes

Relationships:
        Equality: "Sum of X's"
        Parameters: X1 X2 X3 X4 Xsum ;
        Expression:
                { Xsum = X1 + X2 + X3 + X4; }
        Computes: Xsum
        Expression:
                { X4 = Xsum - (X1 + X2 + X3); }
        Computes: X4
        EndEquality
EndRelationships
EndInputSpec
```

Figure B.3. PistonInputs.inputspec - defining the input spec of Piston entity.

```
OutputSpec: PistonOutputs
FilterCommand: "awk -f /home/leeh/Frodo-piston/outputs.awk \
                  < $$FTMPDIR/piston.outdata"
Attributes:
    Category: FilterCommand
        Integer: ExitStatus
        Default: 0
        EndInteger
    EndCategory
        Numeric: G1
        Oom: 10
        EndNumeric
        Numeric: G2
        Oom: 10
        EndNumeric
        Numeric: G3
        Oom: 10
        EndNumeric
        Numeric: G4
        Oom: 10
        EndNumeric
        Numeric: Volume
        Oom: 10
        EndNumeric
EndAttributes
EndOutputSpec
```

Figure B.4. PistonOutputs.outputspec - defining the output spec of Piston entity.

The module entity, Demo, contains only two trivial computations, with three parameters defined, A, B, and C. Figure B.5 shows the module definition in MSAT script.

A = 2 * B
C = 3 * sqrt(A)

The link entity, Piston_Demo, establishes the data dependency between the Piston program and Demo module. For illustration, a simple dependency is defined as

X1 = A
X2 = B

Figure B.6 shows the link definition in MSAT script.

```
Module: Demo

Attributes:
        Numeric: A
        Dom: 10
        EndNumeric

        Numeric: B
        Dom: 20
        EndNumeric

        Numeric: C
        Dom: 10
        EndNumeric
EndAttributes

Relationships:
        Equality: "First Equality"
        Parameters: A B ;
        Expression: { A = 2 * B; }
        Computes: A
        EndEquality

        Equality: "Second Equality"
        Parameters: A C ;
        Expression:
                { C = 3*sqrt(A); }
        Computes: C
        EndEquality
EndRelationships

EndModule
```

Figure B.5. Demo.module - defining the module entity Demo.

```
Link: Piston_Demo
Linkages:
        Program2 isA Piston
        Module3 isA Demo
EndLinkages
Relationships:
        Equality: "X1 Equality"
        Parameters: Program2.Inputs.X1 Module3.A ;
        Expression: { Program2.Inputs.X1 = Module3.A ; }
        Computes: Program2.Inputs.X1
        EndEquality

        Equivalence: "X2 Equivalence"
        Parameters: Program2.Inputs.X2 Module3.B ;
        EndEquivalence
EndRelationships
EndLink
```

Figure B.6. Piston_Demo.link - defining the link entity Piston_Demo.

Running Example

To run the example problem, first make sure that the directory that contains the entity definition files is scanned when MSAT starts up. This is done by including a scan command below in the $HOME/.frodorc:

```
scan "/your/directory/name/example/"
```

When the library window comes up, the canvas should contains the entities defined in the example directory. Proceed with the Part A descriptions to define the model. E.g., "File -> New" and then instantiate the three entities to the model, then bring up entity window for the input/output spec of Piston entity, and module entity window of Demo. Changing values of X1, X2, X3 and X4 will invoke the execution of the Piston and Demo entities. Figure B.7 shows the sample screen dump of running this application.

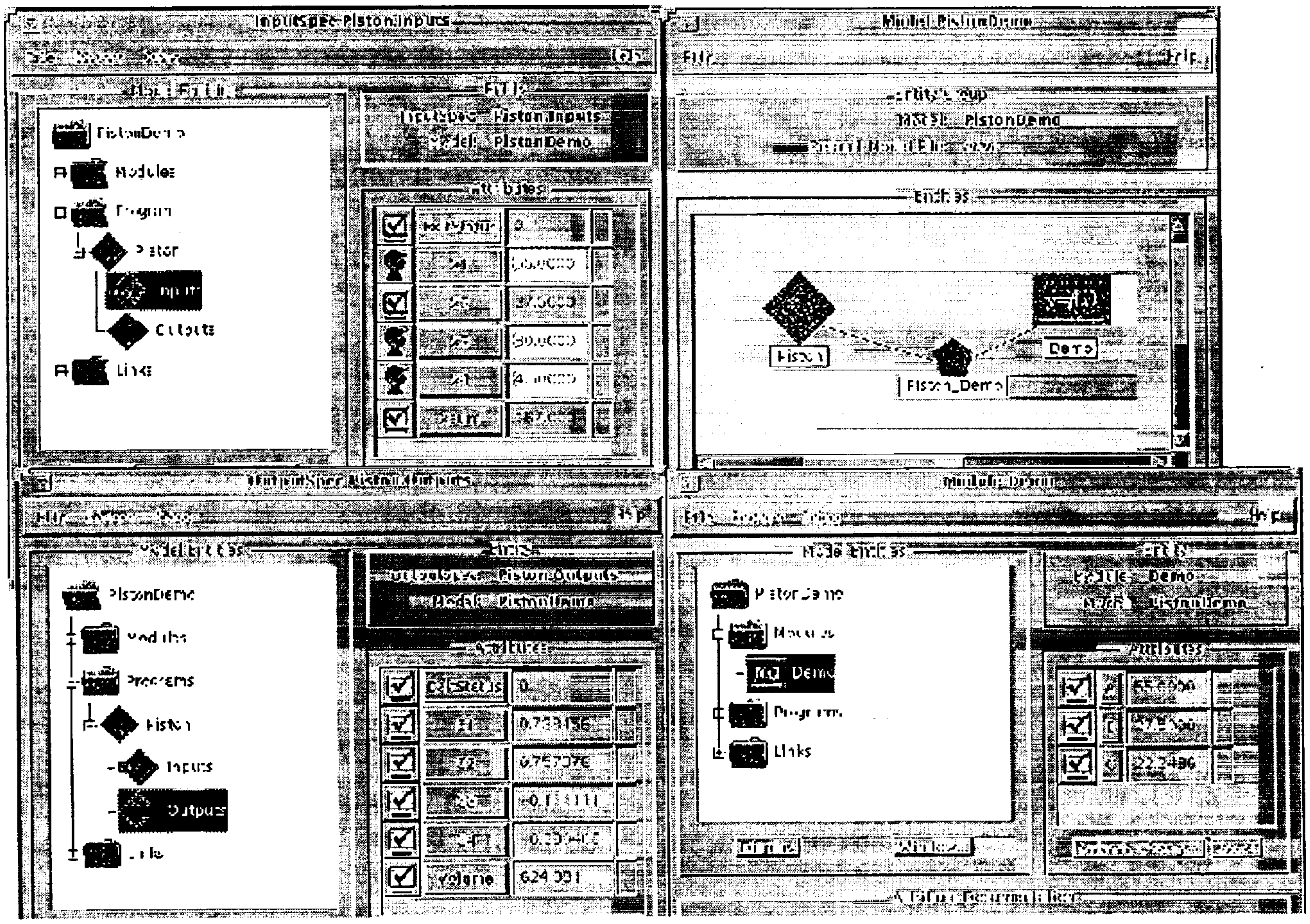


Figure B.7. Running the Example Problem

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** November 1998	**3. REPORT TYPE AND DATES COVERED** Final Contractor Report

4. TITLE AND SUBTITLE Mechanical System Analysis/Design Tool (MSAT) Quick Guide	**5. FUNDING NUMBERS** WU-509-10-31-00 NAS3-26617
6. AUTHOR(S) HauHua Lee, Mark Kolb, and Jack Madelone	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Engineering Mechanics Laboratory GE Corporate Research and Development Center Schenectady, New York	**8. PERFORMING ORGANIZATION REPORT NUMBER** E-11438
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135-3191	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER** NASA CR—1998-208684

11. SUPPLEMENTARY NOTES

Responsible person, Charles Lawrence, Structures and Acoustics Division, NASA Lewis Research Center, organization code 5900, (216) 433-6048.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified - Unlimited Subject Category: 31 Distribution: Nonstandard This publication is available from the NASA Center for AeroSpace Information, (301) 621-0390.	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(Maximum 200 words)***

MSAT is a unique multi-component multi-disciplinary tool that organizes design analysis tasks around object-oriented representations of configuration components, analysis programs and modules, and data transfer links between them. This creative modular architecture enables rapid generation of input stream for trade-off studies of various engine configurations. The data transfer links automatically transport output from one application as relevant input to the next application once the sequence is set up by the user. The computations are managed via constraint propagation – the constraints supplied by the user as part of any optimization module. The software can be used in the preliminary design stage as well as during the detail design of product development process.

14. SUBJECT TERMS Air breathing engines; Design; Software tools	**15. NUMBER OF PAGES** 21
	16. PRICE CODE A03

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT**